

**МИНОБРНАУКИ РОССИИ**

**федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технологический университет «СТАНКИН»**

**(ФГБОУ ВО «МГТУ «СТАНКИН»)**

|  |
| --- |
|  |

|  |  |
| --- | --- |
| **Институт**  **цифровых**  **интеллектуальных**  **систем** | **Кафедра**  **компьютерных**  **систем управления** |

**Бакулев Халил Садыкович**

**«Исследование** **применения OpenCASCADE для задач трёхмерного моделирования в машиностроении»**

**Выпускная квалификационная работа по направлению**

**15.04.04 «Автоматизация технологических процессов и производств»**

|  |  |  |
| --- | --- | --- |
| Зав. кафедрой,  д.т.н., профессор | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Мартинов Г.М. |
| Руководитель,  д.т.н., профессор | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Пушков Р.Л. |
| Студент гр. АДМ-22-04 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Бакулев Х.С. |

Москва

2023

**Оглавление**

[**Введение** 3](#_Toc138341103)

[**Глава 1. Анализ существующих программных средств для реализации модуля визуализации 3D объектов** 4](#_Toc138341104)

[**1.1.** **Анализ средств разработки модуля визуализации** 4](#_Toc138341105)

[**1.2.** **Анализ существующих средств визуализации** 4](#_Toc138341106)

[**1.3.** **Обзор используемых средств разработки** 6](#_Toc138341107)

[**Глава 2. Построение структуры модуля визуализации** 9](#_Toc138341108)

[**2.1.** **Основные концепции проектирования** 9](#_Toc138341110)

[**2.2.** **Описание отношений классов и их описание** 11](#_Toc138341111)

[**Глава 3. Программная реализация приложения** 12](#_Toc138341112)

[**3.1.** **Подключение библиотеки OpenCascade** 12](#_Toc138341113)

[**3.2.** **Описание реализации классов модуля визуализации** 14](#_Toc138341114)

[**Выводы** 25](#_Toc138341115)

[**Заключение** 26](#_Toc138341116)

[**Список литературы** 27](#_Toc138341117)

# **Введение**

При разработке приложения системы управления робототехническом комплексом (далее РТК) для расчёта траектории движения РТК в реальном времени необходима реализация математической модели, учитывающая кинематические и динамические параметры системы. Однако, наличие математической модели недостаточно, так как необходима визуализация рассчитанных траекторий для контроля выбранной оптимальной траектории перед началом обработки заготовки в РТК.

Для визуализации работы системы управления РТК можно использовать готовые CAD/CAM системы, такие как: T-FLEX, SolidWorks, AutoCAD и тд. Однако, использование готовых CAD/CAM систем невозможно при разработке собственных систем управления РТК, так как готовые CAD/CAM системы “тяжеловесны”, что усложняет или делает невозможным интеграцию системы управления РТК внутри существующих CAD/CAM систем.

Приложение управления РТК должно являться системой реального времени для своевременной реакции на показания датчиков и расчёта динамических и кинематических параметров системы. Включая данные ограничения, решением описанной задачи является разработка собственного модуля приложения для визуализации рассчитанных траекторий.

В данной работе рассматриваются этапы разработки модуля приложения управления РТК для визуализации и анимации 3D объектов с выполнением ранее описанных требований.

**Цель работы:**

Применения OpenCASCADE для задач трёхмерного моделирования в машиностроении (в системе управления РТК).

**Задачи:**

1. Провести анализ существующих средств разработки.
2. Построить структуру модуля визуализации.
3. Программного реализовать модуль визуализации.

# **Глава 1. Анализ существующих программных средств для реализации модуля визуализации 3D объектов**

## **Анализ средств разработки модуля визуализации**

Перед началом анализа существующих средств визуализации, необходимо рассмотреть существующие средства разработки приложения. Разработка приложения будет вестись на одном из языков программирования, критерием выбора которого является скорость выполнения приложения и скорость разработки. Группа языков обладающие наибольшей скоростью выполнения – компилируемые языки программирования. В отличии от интерпретируемых языков программирования код, написанный на компилируемом языке, преобразуется в команды процессора для конкретного семейства исполняющих устройств. Данный подход к разработке усложняет реализацию кроссплатформенных приложений, но так как данной задачи в работе не стоит, этот недостаток не учитывается в работе.

Ниже приведены примеры одни из самых популярных языков программирования входящие в различные группы [1].

Примеры компилируемых языков:

* C;
* C++;
* Pascale;
* Go.

Приме ры не компилируемые языков:

* PHP;
* Perl;
* Ruby;
* Python.

В данной работе будет использоваться язык программирование С++. Программы, написанные на данном языке, показывают высокую скорость выполнения [2]. Существует большое количество OpenSource библиотек, написанных на С++, что ускоряет разработку приложения. Разработка приложения управления РТК производится на данном языке. При выборе иного языка программирования, модуль пришлось бы реализовывать в виде внешне подключаемого программного модуля, что вводило бы дополнительные ограничения в разработке и усложняло бы процесс разработки.

* 1. **Анализ существующих средств визуализации**

В большинстве существующих САПР системах используется спецификация OpenGL для визуализации 3D объектов на сцене. Однако, OpenGL не имеет структурного API для описания сцены, а ее интерфейс предоставляет лишь базовые средства для трехмерного рендеринга. Это весьма затрудняет применение API OpenGL без использования вспомогательного кода, который обычно реализуется собственными силами и требует много времени и ресурсов [3]. Для более эффективной разработки приложения необходимо рассмотреть высокоуровневый библиотеки с поддержкой удобного API.

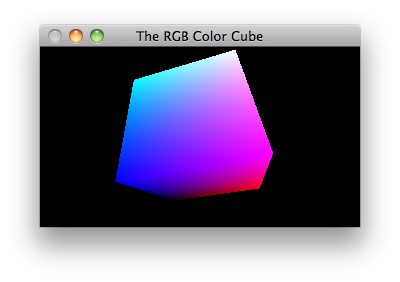


Рисунок 1. Пример визуализации 3D объекта с использование спецификации OpenGL на языке программирования С++.

Отдельных библиотек для визуализации 3D объектов найдено не было, но существуют геометрические ядра с графическими модулями. Таким ядром является C3D Toolkit от компании C3D Labs. C3D Toolkit - специализированный инструмент разработки программного обеспечения (SDK), отвечающий за построение, редактирование, визуализацию и конвертацию геометрических моделей [4]. Данный продукт является полноценным математическим ядром для построения САПР приложений, но данная библиотека является проприетарная, что не позволяет её использовать в нашем модуле.

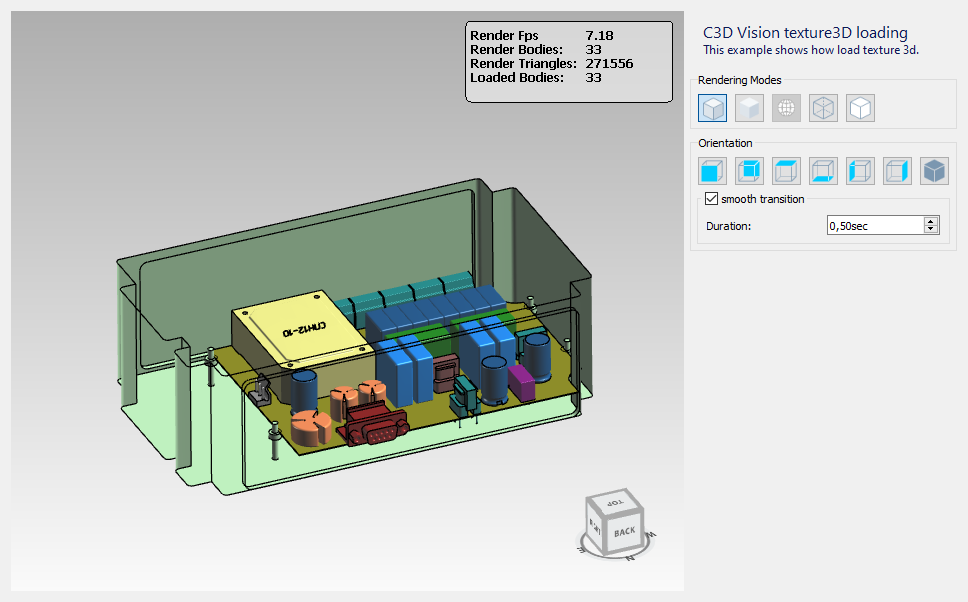


Рисунок 2. Приложение, разработанное с использованием C3D Toolkit.

Открытым геометрическим ядром является OpenCascade. OpenCascade – продукт, сочетающий в себе набор библиотек и средств разработки ПО, ориентированного на 3D-моделирование, в особенности систем автоматизированного проектирования (САПР). Начиная с версии 6.7.0, исходный код доступен и распространяется по лицензии GNU LGPL 2.1 [5]. Более подробно библиотека Open Cascade описана в следующем разделе.

* 1. **Обзор используемых средств разработки**

Open CASCADE Technology – это объектно-ориентированная библиотека классов C++, предназначенная для быстрого создания сложных прикладных приложений CAD/CAM/CAE. Библиотека OCCT разработана для работы с двух- или трехмерным (2D или 3D) геометрическими моделями в универсальных или специализированных системах автоматизированного проектирования (САПР), приложениях для производства или анализа, приложениях для моделирования или инструментах для визуализации [6].

Библиотека OCCT спроектирована модульной, структура показана на рисунке ниже.

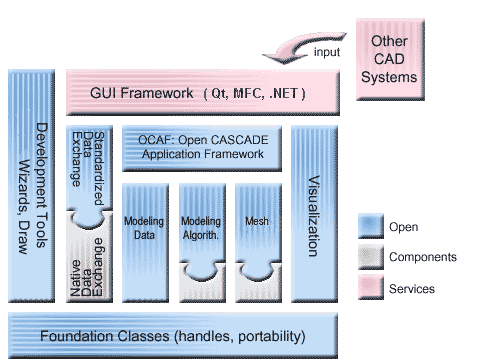


Рисунок 3. Модульная структура библиотеки OpenCascade.

Описание основных модулей:

* Foundation Classes – модуль в классах которого определенны основные типы данных, методы и константы, используемые зависящих модулях;
* Modeling Data – модуль, моделирования элементы которого предоставляет структуры данных для представления 2D- и 3D-геометрических примитивов и их композиций в моделях САПР;
* Modeling Algorithms – модуль, реализующий широкий набор геометрических и топологических алгоритмов;
* Visualization - модуль реализующий механизмы графического представления данных и их отображения;
* Data Exchange – модуль, реализующий импорт и экспорт данных из популярных форматов различных CAD систем;
* Application Framework – модуль, реализующий готовые решения для обработки специфических данных (атрибуты пользователя) и часто используемых функций (сохранение/восстановление, отмена/повтор, копирование/вставка, отслеживание изменений САПР и т. д.).

Классы C++ и другие типы сгруппированы в пакеты. Пакеты организованы в наборы инструментов (библиотеки), которые возможно импортировать в собственное приложение.

Для реализации графического интерфейса модуля приложения управления РТК используется фреймворк Qt. Qt – это фреймворк для разработки кроссплатформенного программного обеспечения на языке программирования C++. Qt позволяет запускать написанное с его помощью программное обеспечение в большинстве современных операционных систем путём простой компиляции программы для каждой системы без изменения исходного кода. Включает в себя все основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами для работы с сетью, базами данных и XML. Является полностью объектно-ориентированным, расширяемым и поддерживающим технику компонентного программирования.

Отличительная особенность — использование метаобъектного компилятора — предварительной системы обработки исходного кода. Расширение возможностей обеспечивается системой плагинов, которые возможно размещать непосредственно в панели визуального редактора. Также существует возможность расширения привычной функциональности виджетов, связанной с размещением их на экране, отображением, перерисовкой при изменении размеров окна.

Для более удобной работы с фреймворком Qt, для разработки приложения использовался IDE Qt Redactor (рис. 4).

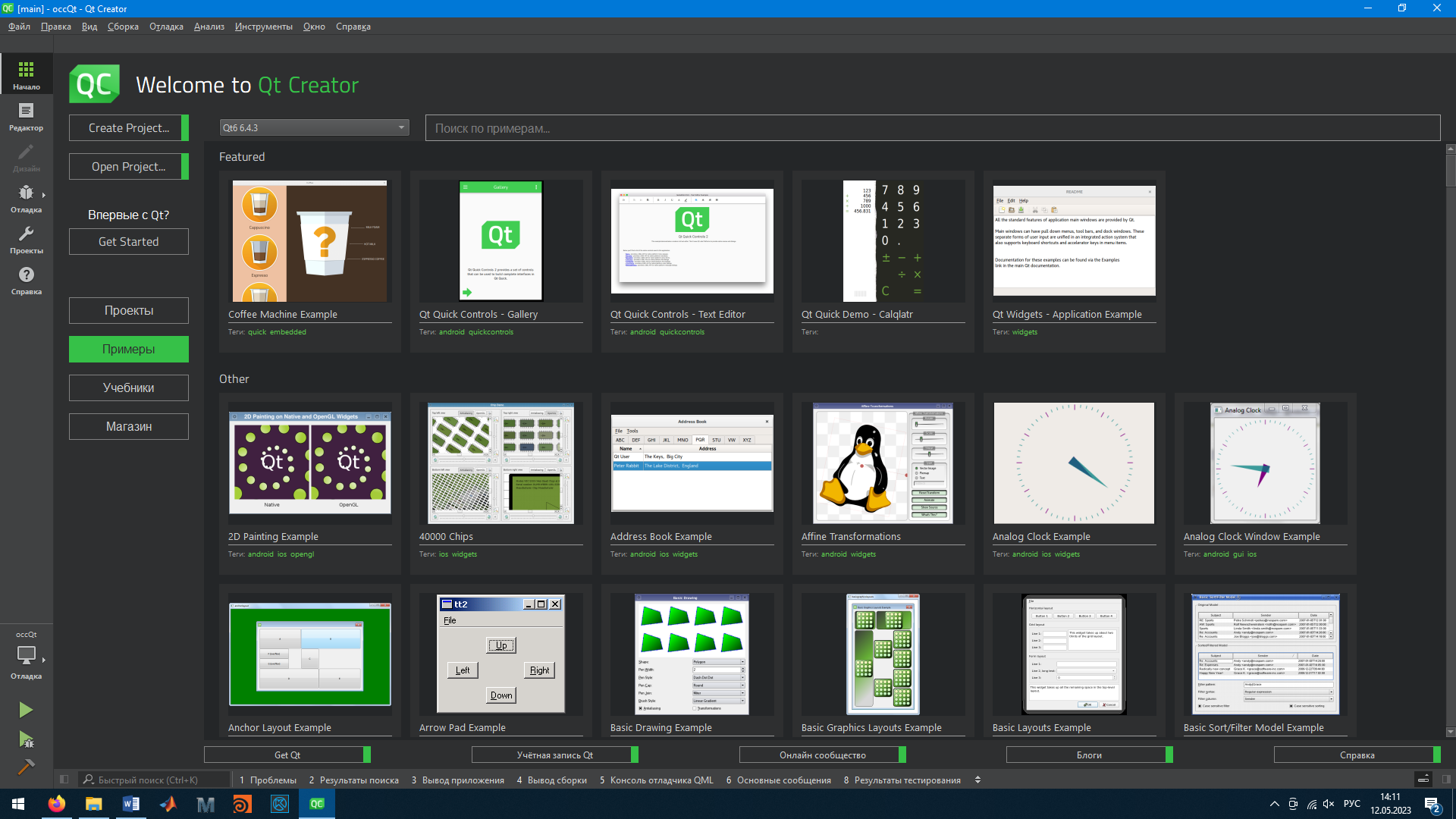


Рисунок 4. Внешний вид Qt Redactor.

В Qt Redactor возможно использовать системы сборок: cmake, qmake, Qbc. При разработке модуля будет использоваться система сборки qmake, так как она обладает более простым синтаксисом относительно cmake, а также автоматизирует многие процессы сборки проекта при работе в Qt Creator.

Qt Redactor включает модуль Qt designer. Qt designer составная часть редактора Qt в котором возможно проектирование графического интерфейса, результатом работы данного модуля являются файлы реализации, объявления С++ классов и файлы с расширением ui в которых находится разметка виджетов в формате XML.

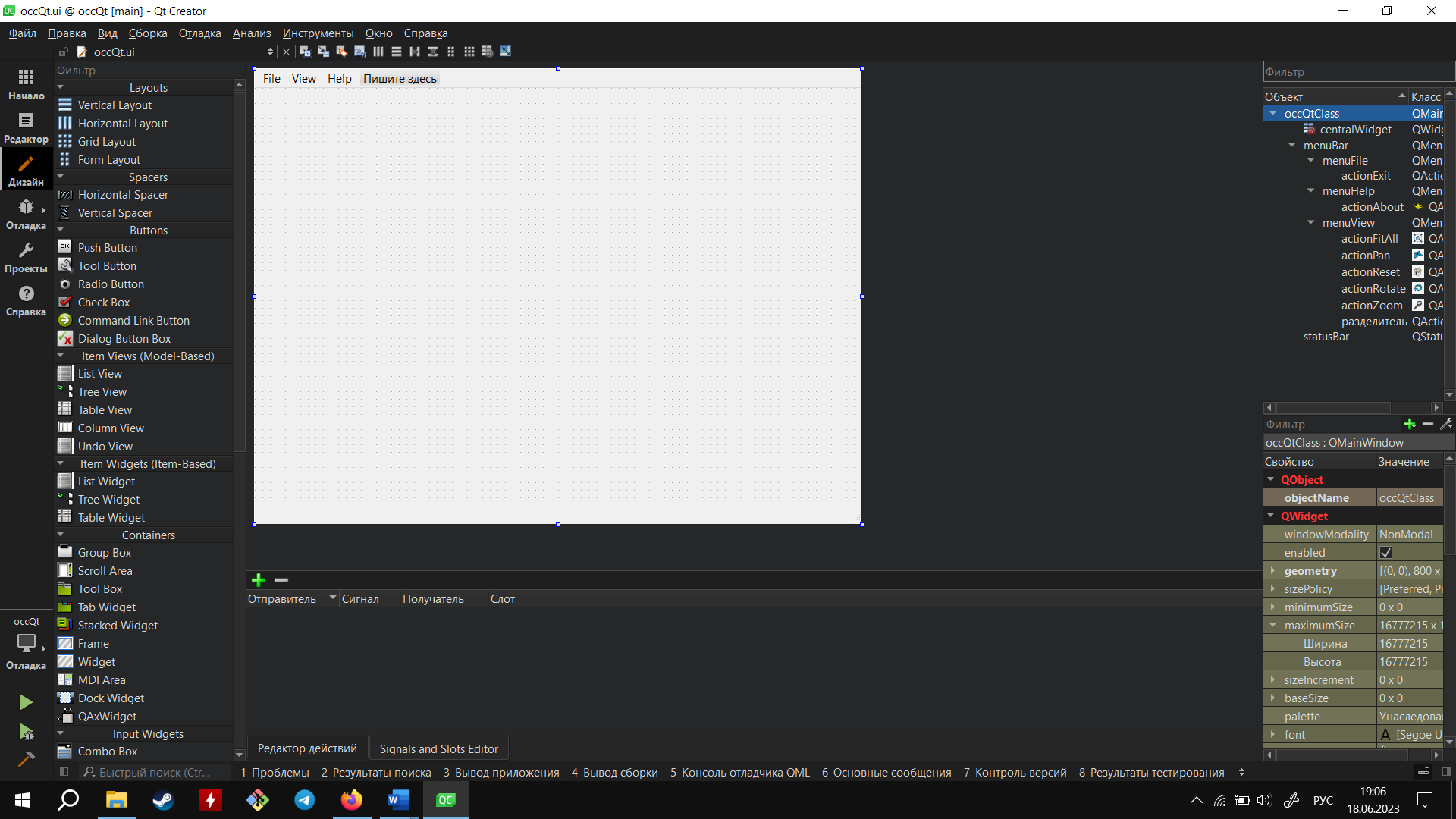


Рисунок 5. Внешний вид Qt Designer.

# **Глава 2. Построение структуры модуля визуализации**

1. 1. **Основные концепции проектирования**

Для выполнения требований к производительности приложения используется подход параллельного или многопоточного\* (далее конкурентного) программирования. Данный принцип разработки приложения подразумевает разбиение логики приложения на независимые потоки, использующие общее пространство данных приложения.

**Поток выполнения** — наименьшая единица обработки, исполнение которой может быть назначено ядром операционной системы. Реализация потоков выполнения и процессов в разных операционных системах отличается друг от друга, но в большинстве случаев поток выполнения находится внутри процесса. Несколько потоков выполнения могут существовать в рамках одного и того же процесса и совместно использовать ресурсы, такие как память.

Потоки могут выполняться как параллельно при наличии достаточно количества ядер процессора, так и псевдопараллельно, то есть выполнения каждого потока происходит короткий промежуток времени, что создаёт у пользователя ощущение параллельного исполнения потоков (рисунок 6). При выполнении разных потоков на одном ядре процессора, учитывается время смены контекста (сохранения состояния одного потока и загрузка состояния другого).

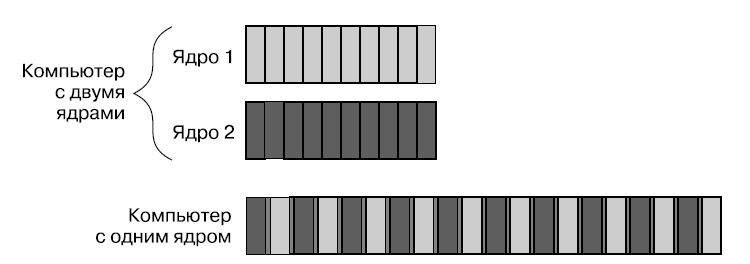


Рисунок 6. Графическое представление конкурентного выполнения приложения с двумя потоками. Светло-серый и темно-серый – потоки.

\*Термины не являются синонимами, в данном контексте их использование как синонимы приемлемо. (подробнее см. [7]).

Реализация конкурентного программирования возможна с использованием:

* Набор библиотек Boost
* OpenMP
* OpenThreads
* POCO Thread (часть проекта POCO)
* Pthreads (Ptreads-w32)
* Zthread
* Qt Threads
* Intel Threading Building Blocks
* Стандартная библиотека STL (C++11).

При разработке приложения на языке С++, предпочтительнее использовать функционал стандартной библиотеки STL, так как стандарт языка С++ поддерживает всеми компиляторами. Стандарт гарантирует одинаковый результат функционала библиотеки вне зависимости от компилятора или платформы разработки. Однако, пользуясь функционалам Qt и метаобъектным компилятором, удобнее использовать встроенный стредства фреймворка Qt. В Qt концепция конкурентного программирования реализовано с помощью класса QThread.

Для реализации многопоточности в собственном приложении, необходимо создать класс, унаследованный от класса QThread и в нём переопределить виртуальный метод *void run()*. При запуске данного метода, будет создан новый поток, который будет «жить» до окончания метода *void run()*. Таким образом класс QThread управляет созданием, завершением и передачей аргументов новому потоку. Внутри каждого потока возможен вызов собственного цикла обработчика событий Qt. Каждый поток в Qt может иметь приоритет исполнения.

Существует восемь уровней приоритета исполнения:

1. QThread::IdlePriority - запланировано только тогда, когда никакие другие потоки не запущены.
2. QThread::LowestPriority - запланировано реже, чем LowPriority.
3. QThread::LowPriority - запланировано реже, чем NormalPriority.
4. QThread::NormalPriority - приоритет для операционной системы по умолчанию.
5. QThread::HighPriority - запланировано чаще, чем NormalPriority.
6. QThread::HighestPriority - запланировано чаще, чем HighPriority.
7. QThread::TimeCriticalPriority - планируется как можно чаще.
8. QThread::InheritPriority - использовать тот же приоритет, что и создающий поток. Это значение по умолчанию.

Во фреймворке Qt используется подход «сигналов и слотов».

**Сигналы и слоты** — подход, используемый в некоторых языках программирования и библиотеках который позволяет реализовать шаблон «наблюдатель», минимизируя написание повторяющегося кода. Концепция заключается в том, что компонент может посылать сигналы, содержащие информацию о событии. В свою очередь другие компоненты могут принимать эти сигналы посредством специальных функций — слотов.

Шаблон программирование «наблюдатель» заключается в реализации у класса механизма, который позволяет объекту этого класса получать оповещения об изменении состояния других объектов и тем самым наблюдать за ними.

* 1. **Описание отношений классов и их описание**

Приложение состоит из трёх классов.В данном описание не будут рассматриваться классы, созданные автоматически Qt designer, так как в них реализован функционал для элементов разметки ui файлов. Файлы созданные Qt designer не учувствуют в логике управления приложения, а отвечаю только за графическое отображение и функционирование элементов виджетов.

Ниже приведена UML диаграмма отношений классов.

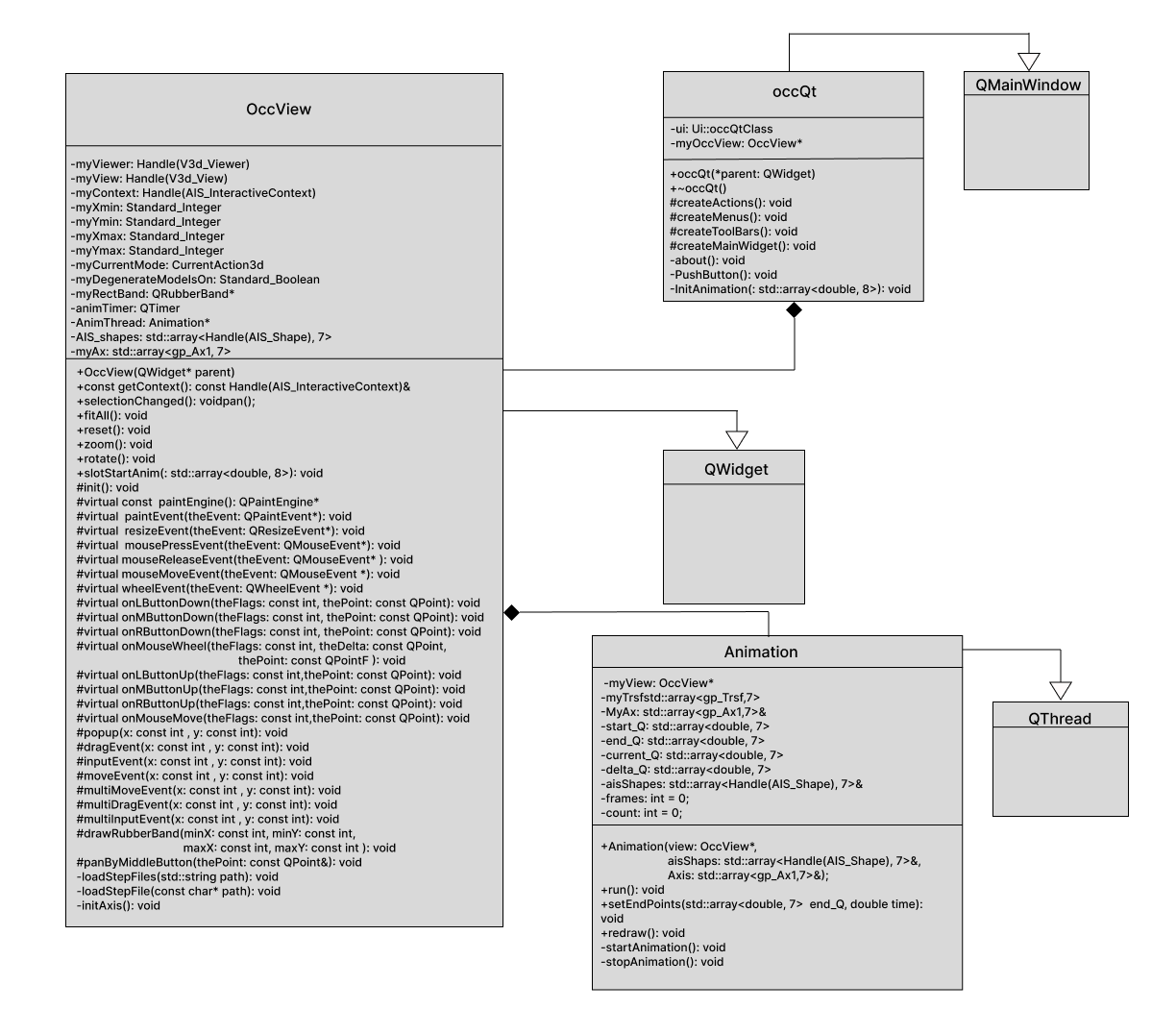


Рисунок 7. Диаграмма UML классов.

Отношение между классами описываются композицией в терминологии UML. Ниже приведено функциональное описание классов:

occQt – класс реализующий главное окно приложения, внутри данного класса находится указатель на объект класса OccView.

OccView – класс реализующий виджет отображения 3D объектов, загрузку моделей 3D объектов в формате step, методы изменяющие вид камеры сцене. Внутри класса находится указатель на объект класса Animation.

Animation – класс создающий поток выполнения в котором происходит перерисовка анимации. Данному классу передаются параметры анимируемых элементов и параметры анимации.

# **Глава 3. Программная реализация приложения**

* 1. **Подключение библиотеки OpenCascade**

Библиотека OpenCascade распространяется в двух видах: в готовом скомпилированном формате; в формате исходного кода для дальнейшей компиляции. При самостоятельной сборки библиотеки необходимо скачать/собрать зависимые библиотеки и указать пути расположения до библиотек в системе сборки CMake.

Для упрощения и ускорения разработки модуля приложения в данной работе используется скомпилированная библиотека OpenCascade. Разработчиками библиотека OpneCascade скомпилированная собрана с использованием компилятора MSVC. Для дальнейшей работы с библиотекой необходимо использовать в Qt Redactor компилятор MSVC, стандартный компилятор в IDE – MinGW.

Выше было сказано, что в данном модуле приложения используется система сборки qmake. В системе сборке qmake описания проекта находится в файле с расширением \*.pro. Ниже приведен листинг occQt.pro – файла описания проекта модуля приложения визуализации.

QT       += widgets core gui opengl openglwidgets #Подключение модулей Qt используемых в приложении

TARGET = occQt #Название цели (приложения)

TEMPLATE = app #Тип приложения (библиотека, приложение или плагин)

SOURCES += main.cpp \ # \*.cpp файлы используемые в проекте

    myanimation.cpp \

    occQt.cpp       \

    occView.cpp

CONFIG += c++11 # дополнительные параметры (стандарт языка)

HEADERS  += \ # используемые в проекте \*.h файлы

    myanimation.h \

    occQt.h \

    occView.h

FORMS    += \ # используемые формы

    occQt.ui

RESOURCES += \ # папка, содержащая ресурсы проекта(иконки)

    .qrc

# Переменная содержащая путь до библиотеки OpenCascade

CASROOT = "D:\\Programs\\OpenCASCADE-7.7.0-vc14-64\\opencascade-7.7.0"

INCLUDEPATH += $${CASROOT}/inc #Подключение \*.h файлов библиотеки Open Cascade

LIBS += -L$${CASROOT}/win64/vc14/lib/ #Подключение пути до статических библиотек \*.lib Open Cascade

#Подключение библиотек Open Cascade перечисленных ниже

LIBS +=         \

    -lTKernel   \

    -lTKMath     \

    -lTKG3d     \

    -lTKBRep     \

    -lTKGeomBase \

    -lTKGeomAlgo \

    -lTKTopAlgo \

    -lTKPrim     \

    -lTKBO       \

    -lTKBool     \

    -lTKOffset   \

    -lTKService \

    -lTKV3d     \

    -lTKOpenGl   \

    -lTKFillet   \

    -lTKernel \

    -lTKIGES \

    -lTKSTL \

    -lTKVRML \

    -lTKSTEP \

    -lTKSTEPAttr \

    -lTKSTEP209 \

    -lTKSTEPBase \

    -lTKG2d \

    -lTKXSBase \

    -lTKShHealing\

    -lTKHLR \

    -lTKMesh \

    -lTKCDF \

    -lTKXDESTEP \

Листинг 1. Файл occQt.pro

* 1. **Описание реализации классов модуля визуализации**

В данной главе будут рассматриваться основные элементы реализации классов проекта. Проект расположен на сервисе github по ссылке <https://github.com/HalilProgr/OccKukaAnimation>. В данном репозитории находятся зависимые файлы (кроме используемых библиотек), включая step модели.

При описании классов будут представлены \*.h файлы и листинги важных методов классов.

**Описание реализации класса occQt**

Для упрощения описания класса ниже приведен рисунок главного окна приложения.

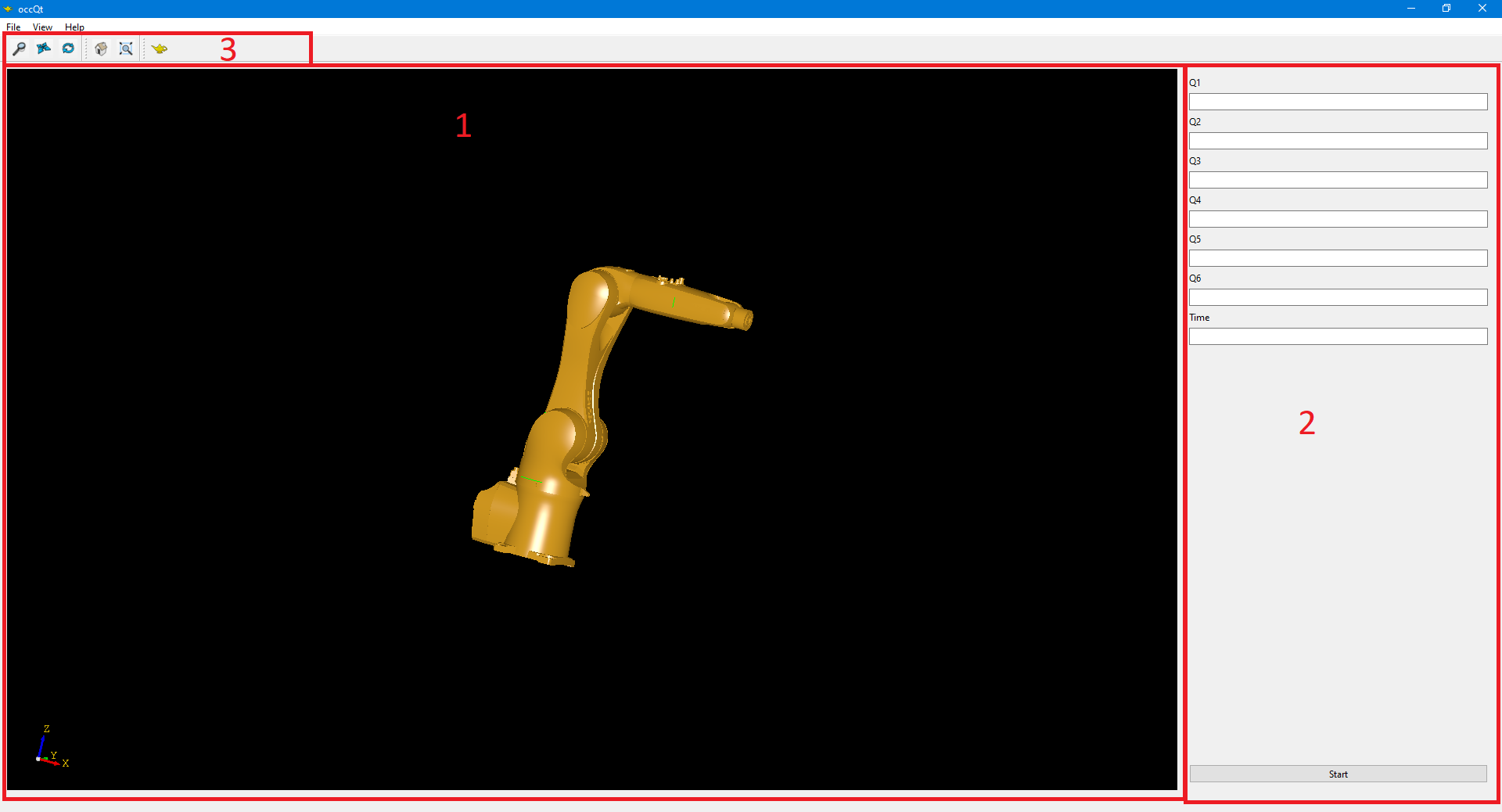


Рисунок 8. Внешний вид главного окна. 1 – виджет OccView 2 – виджет ввода данных и запуска анимации 3 – ToolBar.

Класс occQt состоит из трёх частей:

1. виджет (1 на рис.8) отображения и анимации 3D объектов. При наведении мышки на данный виджет возможен поворот камеры, перемещение по сцене и изменение масштаба.
2. виджет (2 на рис. 8) запуска анимации, в данном виджете расположены поля ввода и кнопка запуска анимации. В поля записываются желаемые обобщённые координаты манипулятора. При нажатии кнопки испускается сигнал в класс OccView (2 на рис 8).
3. Tool Bar (3 на рис. 8) – в данной части реализован функционал переключения состояния указателя мыши, переход в начало координат, масштабирование сцены относительно визуализированных объектов.

#ifndef OCCQT\_H

#define OCCQT\_H

// подключение \*.h класса, созданного Qt Designer

#include "ui\_occQt.h"

// подключение \*.h файлов библиотеки OpenCascade

#include <AIS\_InteractiveContext.hxx>

#include <AIS\_Shape.hxx>

// подключение \*.h файлов библиотеки Qt

#include <QLabel>

#include <QLineEdit>

#include <QPushButton>

#include <QTimer>

#include <QDebug>

#include <QHBoxLayout>

class OccView; // Объявление класса использующегося в данном классе.

class occQt : public QMainWindow

{

    Q\_OBJECT    // Макрос для работы метообъектного компилятора Qt

public:

    // Конструктор и деструктор

    occQt(QWidget \*parent = nullptr);

    ~occQt();

protected:

    // Метод инициализирующий соединение сигналов и слотов классов

    void createActions();

    // Метод инициализирующий Tool Bar

    void createToolBars();

    // Метод инициализирующий объект отображения

    void createMainWidget();

private slots:

    // Слот срабатывающий при нажатии About в ToolBar

    void about();

    // Слот, срабатывающий на сигнал нажатия кнопки в UI интерфейсе, считывает значения полей и отправляет данные в сигнале InitAnimation

    void PushButton();

signals:

    void InitAnimation(std::array<double, 8>);

private:

    // объект UI отображения созданный Qt Designer

    Ui::occQtClass ui;

    // Указатель, содержащий виджет окна отображения

    OccView\* myOccView;

};

#endif // OCCQT\_H

Листинг 2. Объявление класса occQt.

**Описание реализации класса OccView**

В библиотеке OpenCascade графическое отображение разделено от математического описания топологии объектов сцены с помощью модуля – AIS (Application Interactive Services). Внутри модуля AIS определён класс представления фигур – AIS\_shape. Таким образом топология объектов OpenCascade должна быть преобразована в AIS представление для дальнейшего отображения.

Загрузка 3D моделей в формате step осуществляется объектом класса STEPControl\_Reader. Step файл конвертируется в объект TopoDS\_Shape – объект представления 3D объектов в OpenCascade. Для визуального отображения объекты TopoDS\_Shape конвертируются в AIS\_shape.

#ifndef \_OCCVIEW\_H\_

#define \_OCCVIEW\_H\_

// подключение объявления класса Animation

#include "myanimation.h"

// подключение \*.h файлов библиотеки Qt

#include <QOpenGLWidget>

#include <QRubberBand>

#include <QTimer>

// подключение \*.h файлов библиотеки OpenCascade

#include <AIS\_InteractiveContext.hxx>

#include <AIS\_Shape.hxx>

class Animation;

class QMenu;

class QRubberBand;

class OccView : public QWidget

{

    Q\_OBJECT

public:

    // объявление перечисления, описывающие состояние мыши

    enum CurrentAction3d

    {

        CurAction3d\_Nothing,

        CurAction3d\_DynamicZooming,

        CurAction3d\_WindowZooming,

        CurAction3d\_DynamicPanning,

        CurAction3d\_GlobalPanning,

        CurAction3d\_DynamicRotation

    };

public:

    // Конструктор класс

    OccView(QWidget\* parent);

    // метод возвращающий контекст

    const Handle(AIS\_InteractiveContext)& getContext() const;

signals:

    void selectionChanged(void);

public slots:

    // операции для изменения представления

    void pan(void);

    void fitAll(void);

    void reset(void);

    void zoom(void);

    void rotate(void);

    // слот запуска анимации

    void slotStartAnim(std::array<double, 8>);

protected:

    virtual QPaintEngine\* paintEngine() const;

    // Методы перерисовки и масштабирования

    virtual void paintEvent(QPaintEvent\* theEvent);

    virtual void resizeEvent(QResizeEvent\* theEvent);

    // Методы связанные с мышью

    virtual void mousePressEvent(QMouseEvent\* theEvent);

    virtual void mouseReleaseEvent(QMouseEvent\* theEvent);

    virtual void mouseMoveEvent(QMouseEvent \* theEvent);

    virtual void wheelEvent(QWheelEvent \* theEvent);

    // Методы связанные с нажатием на клавиши мыши

    virtual void onLButtonDown(const int theFlags, const QPoint thePoint);

    virtual void onMButtonDown(const int theFlags, const QPoint thePoint);

    virtual void onRButtonDown(const int theFlags, const QPoint thePoint);

    virtual void onMouseWheel(const int theFlags, const QPoint theDelta, const QPointF thePoint);

    virtual void onLButtonUp(const int theFlags, const QPoint thePoint);

    virtual void onMButtonUp(const int theFlags, const QPoint thePoint);

    virtual void onRButtonUp(const int theFlags, const QPoint thePoint);

    virtual void onMouseMove(const int theFlags, const QPoint thePoint);

protected:

    // Метод инициализации главного окна отображения

    void init(void);

    // Перемещение по методы осуществляющие перемещение по сцене

    void popup(const int x, const int y);

    void dragEvent(const int x, const int y);

    void inputEvent(const int x, const int y);

    void moveEvent(const int x, const int y);

    void multiMoveEvent(const int x, const int y);

    void multiDragEvent(const int x, const int y);

    void multiInputEvent(const int x, const int y);

    void drawRubberBand(const int minX, const int minY, const int maxX, const int maxY);

    void panByMiddleButton(const QPoint& thePoint);

private:

    // Метод инкапсуляции загрузки step файлов

    void loadStepFiles(std::string path);

    // Метод загрузки одного step файла

    void loadStepFile(const char\* path);

    // Метод инициализации объектов осей поворота отображаемых объектов

    void initAxis();

    //Handle -  умный указатель из библиотеки OCCT

    // V3d\_Viewer - класс обёртка над графическим драйвером OpenGL

    Handle(V3d\_Viewer) myViewer;

    // V3d\_View - Окно просмотра созданное V3d\_Viewer

    Handle(V3d\_View) myView;

    // AIS\_InteractiveContext - класс отвечающий за визуализацию 3D объектов

    Handle(AIS\_InteractiveContext) myContext;

    // Standard\_Integer переопределенное int в библиотеке OCCT

    // в переменных описанных ниже сохраняются координаты мыши

    Standard\_Integer myXmin;

    Standard\_Integer myYmin;

    Standard\_Integer myXmax;

    Standard\_Integer myYmax;

    // Переменная описывающая текущий режим мыши

    CurrentAction3d myCurrentMode;

    Standard\_Boolean myDegenerateModeIsOn;

    // Объект отображающий прямоугольник выделения

    QRubberBand\* myRectBand;

    //Таймер, отвечающий за отправку сигнала для перерисовки сцены

    QTimer animTimer;

    // Указатель на объект управления анимацией

    Animation\* AnimThread;

    // Массив из указателей на объекты отображения

    std::array<Handle(AIS\_Shape), 7> AIS\_shapes;

    // Массив из указателей на объекты осей повороты отображаемых объектов

    std::array<gp\_Ax1, 7> myAx;

};

#endif // \_OCCVIEW\_H\_

Листинг 3. Объявление класса OccView.

Основными методами класса OccView являются:

1. init() – метод инициализация графического окна.
2. loadStepFile() – метод отвечающий за загрузку и преобразовпания step файла в представлении OpenCascade.
3. slotStartAnim() – слот запускающий анимацию.

В OpenCascade для централизованной работы со сценой реализован класс AIS\_InteractiveContext – класс контекста. Объект класса контекста принимает объект централизованного управления графического отображения типа V3d\_Viewer. Объект класса V3d\_Viewer принимает openGL драйвер конкретной платформы.

void OccView::init()

{

    Handle(Aspect\_DisplayConnection) aDisplayConnection =

            new Aspect\_DisplayConnection();

    if (GetGraphicDriver().IsNull())

    {

        //GetGraphicDriver() - статический метод хранящий в себе указатель на OpenGL драйвер

        GetGraphicDriver() = new OpenGl\_GraphicDriver(aDisplayConnection);

    }

    WId window\_handle = (WId) winId(); // winId - Возвращает идентификатор оконной системы виджета

    // Макрос в котором определяется платформа, на которой запущено приложение и создаётся

    // графический драйвер на основе платформы.

    #ifdef WNT

        Handle(WNT\_Window) wind = new WNT\_Window((Aspect\_Handle) window\_handle);

    #elif defined(\_\_APPLE\_\_) && !defined(MACOSX\_USE\_GLX)

        Handle(Cocoa\_Window) wind = new Cocoa\_Window((NSView \*) window\_handle);

    #else

        Handle(Xw\_Window) wind = new Xw\_Window(aDisplayConnection, (Window) window\_handle);

    #endif

    // V3d\_Viewer - класс из библиотеки OpenCascade принимающий драйвер, отвечает за создание графического окна

    myViewer = new V3d\_Viewer(GetGraphicDriver());

    myView = myViewer->CreateView(); // Инициализация окна

    myView->SetWindow(wind);

    if (!wind->IsMapped()) wind->Map();

    // AISInteractiveContext - класс, управляющий за графическое отображение объектов в OpenCascade

    myContext = new AIS\_InteractiveContext(myViewer);

    // Дополнительные параметры настройки графического окна.

    myViewer->SetDefaultLights();

    myViewer->SetLightOn();

    myView->SetBackgroundColor(Quantity\_NOC\_BLACK);

    myView->MustBeResized();

    myView->TriedronDisplay(Aspect\_TOTP\_LEFT\_LOWER, Quantity\_NOC\_GOLD, 0.08, V3d\_ZBUFFER);

    myContext->SetDisplayMode(AIS\_Shaded, Standard\_True);

}

Листинг 4. Определение метода init().

Для загрузки step файлов в OpenCascade реализован класс STEPControl\_Reader, объект данного класса считывает и преобразовывает step файл в объект класса TopoDS\_Shape.

Так как загрузка каждой модели трудоемкий и вычислительно затратный процес, то загрузка всех моделей производится поэтапно. В данном случаи робот состоит из 7 звеньев, таким образом метод loadStepFile() вызывается 7 раз.

// Метод принимает путь до step файла в формате С

void OccView::loadStepFile(const char\* path){

    // Статическая переменная-счётчик количества загруженных моделей

    static int countAisLoadedShapes = 0;

    // Инициализация объекта управляющего считыванием step файлов

    STEPControl\_Reader Reader;

    // Считываение и проверка на корректное открытие step файла

    if (Reader.ReadFile(path) == IFSelect\_RetDone)

        qDebug() << "step file loaded! WIN WIN WIN";

    // Преобразование загруженного файла в представление OpenCascade и вывод о состоянии преобразования

    qDebug() << Reader.TransferRoots() << " Reader roots transferred.";

    // Создание временного объекта представления 3D модели

    TopoDS\_Shape temp\_shape = Reader.OneShape();

    // Преобразование математического представления модели в AIS\_shape

    AIS\_shapes[countAisLoadedShapes] = new AIS\_Shape(temp\_shape);

    // отображение загруженной модели

    myContext->Display(AIS\_shapes[countAisLoadedShapes],Standard\_False);

    countAisLoadedShapes++;

}

Листинг 5. Определение метода loadStepFile().

В методе slotStartAnim происходит взаимодействие с классом Animation. Подробное описание реализации класса Animation будет дальше.

// Метод принимает значения из виджета объекта occQt, отвечающего за запуск анимации и ввода данных

void OccView::slotStartAnim(std::array<double, 8> inputData){

    // Массив, передаваемый объекту класса Animation, для инициализации конечных обобщенных координат и длительности анимации

    std::array<double, 7> endPointsQ;

    qDebug() << "from occView.cpp with love";

    for(int i = 0; i < 7; i++){

        qDebug() <<  (endPointsQ[i] = inputData[i]);

    }

    qDebug() << inputData[7];

    // Вызов метода инициализации анимации

    AnimThread->setEndPoints(endPointsQ, inputData[7]);

    // Запуск анимации

    AnimThread->start();

}

Листинг 6. Определение метода slotStartAnim().

В главе 2 была описана концепция конкурентного программирования. Важно отметить что все методы (слоты в том числе) после запуска метода run() объекта класса унаследованного от QThread выполняются в новом потоке. Если между потоками происходит обмен сигналами, то они автоматически преобразовываются системой Qt в специальные сообщения.

Для управления частотой кадров используется таймер - QTimer. Объект данного класса испускает сигнал каждую секунды для запуска перерисовки сцены, до окончания анимации. Для работы объектов генерирующие сигналы, необходим запуск цикла выполнения Qt. При запуске данного цикла в дочернем потоке, происходят «фризы» в элементах класса OccView. Для решения данной проблемы таймер, отвечающий за частоту обновления кадров находится в классе OccView.

#ifndef MYANIMATION\_H

#define MYANIMATION\_H

#include "occView.h"

// подключение библиотек OpenCascade

#include <AIS\_InteractiveContext.hxx>

#include <AIS\_Shape.hxx>

// подключение библиотек Qt

#include <QThread>

#include <QDebug>

class OccView;

//  Класс отвечающий за управление потоком анимации и за состояние анимации

class Animation: public QThread{

    Q\_OBJECT

public:

    Animation(OccView\* view, std::array<Handle(AIS\_Shape), 7>& aisShaps,

              std::array<gp\_Ax1,7>& Axis);

    // Переопределенный метод QThread инициализирующий запуск нового потока

    void run();

    // Метод инициализирующий параметры перемещения

    void setEndPoints(std::array<double, 7>  end\_Q, double time);

private:

    // Указатель окна отображения

    OccView\* myView;

    // Матрицы преобразования звеньев манипулятора

    std::array<gp\_Trsf,7> myTrsf;

    // ссылка на координаты осей поворота осей

    std::array<gp\_Ax1,7>& MyAx;

    // Обобщённые координаты

    std::array<double, 7> start\_Q;

    std::array<double, 7> end\_Q;

    std::array<double, 7> current\_Q;

    std::array<double, 7> delta\_Q;

    // Ссылка на массив отображаемый объектов

    std::array<Handle(AIS\_Shape), 7>& aisShapes;

    // Переменная хранения количества кадров анимации

    int frames = 0;

    int count = 0;

signals:

    // сигналы запуска и остановки таймера

    void startAnimation();

    void stopAnimation();

public slots:

    // Метод перерисовки анимации вызываемый сигналом таймера

    void redraw();

};

#endif // MYANIMATION\_H

Листинг 7. Объявление класса Animation.

Основными методами класса Animation являются:

1. run () – метод запуска нового потока.
2. setEndPoints () – метод ввода параметров анимации.
3. redraw () – метод перерисовки кадров анимации.

void Animation::run(){

    // Цикл работающий до завершения отрисвоки всех кадров

    while(count < frames){};

    // Обновление начальных координат манипулятора

    start\_Q = end\_Q;

    // Обнуление счётчика кадрлов

    count = 0;

    // Отправка сигнала завершения работы таймера

    emit stopAnimation();

}

Листинг 8. Определение метода run ().

Листинг метода setEndPoints.

void Animation::setEndPoints(std::array<double, 7>  end\_Q\_points, double time){

// Расчёт приращения изменения координат

    for(int i = 0; i < 7; i++){

        qDebug() << end\_Q\_points[i];

        delta\_Q[i] = (end\_Q\_points[i] - start\_Q[i])/(time \* 60);

    }

// Расчёт количества кадров

    frames = (int)time \* 60;

// Изменение обобщенных координат

    end\_Q = end\_Q\_points;

    current\_Q = start\_Q;

// Отправка сигнала запуска работы таймера

    emit startAnimation();

}

Листинг 9. Определение метода setEndPoints().

Метод redraw производит инициализацию и перемножение матриц преобразования звеньев робота.

void Animation::redraw(){

    // Инициализация матриц преобразований текущими значениями обобщённых координат

    for(int i = 0; i < 7; i++){

        current\_Q[i]+= delta\_Q[i];

        myTrsf[i].SetRotation(MyAx[i], current\_Q[i] \* M\_PI/180);

    }

    // Установка отображаемых фигур в положения, полученные перемножением матриц преобразований

    aisShapes[0]->SetLocalTransformation(myTrsf[0]);

    aisShapes[1]->SetLocalTransformation(myTrsf[0]\*myTrsf[1]);

    aisShapes[2]->SetLocalTransformation((myTrsf[0]\*myTrsf[1])\*myTrsf[2]);

    aisShapes[3]->SetLocalTransformation(((myTrsf[0]\*myTrsf[1])\*myTrsf[2])\*myTrsf[3]);

    aisShapes[4]->SetLocalTransformation((((myTrsf[0]\*myTrsf[1])\*myTrsf[2])\*myTrsf[3])\*myTrsf[4]);

    aisShapes[5]->SetLocalTransformation(((((myTrsf[0]\*myTrsf[1])\*myTrsf[2])\*myTrsf[3])\*myTrsf[4])\*myTrsf[5]);

    aisShapes[6]->SetLocalTransformation((((((myTrsf[0]\*myTrsf[1])\*myTrsf[2])\*myTrsf[3])\*myTrsf[4])\*myTrsf[5])\*myTrsf[6]);

    myView->getContext()->LastActiveView()->Redraw();

    count++;

}

Листинг 10. Определение метода redraw().

# **Выводы**

В работе был проведен анализ существующих средств разработки модуля визуализации приложения управления РТК. Был выбран язык программирования С++, проанализированы существующее библиотеки для работы с 3D моделями из которых была выбрана библиотека OpenCascade. Для более удобной разработки использовался IDE Qt Creator.

Была разработана структура приложения в виде UML диаграммы с описанием функционала классов. В работе были приведены основные концепции используемые при разработке приложения.

Была реализована программная реализация модуля визуализации приложения управления РТК. Ниже приведены рисунки, отображающие состояние манипулятора при анимации перемещения на обобщённые координаты q1 = 180; q2 = 30, за время t = 3 секунды.

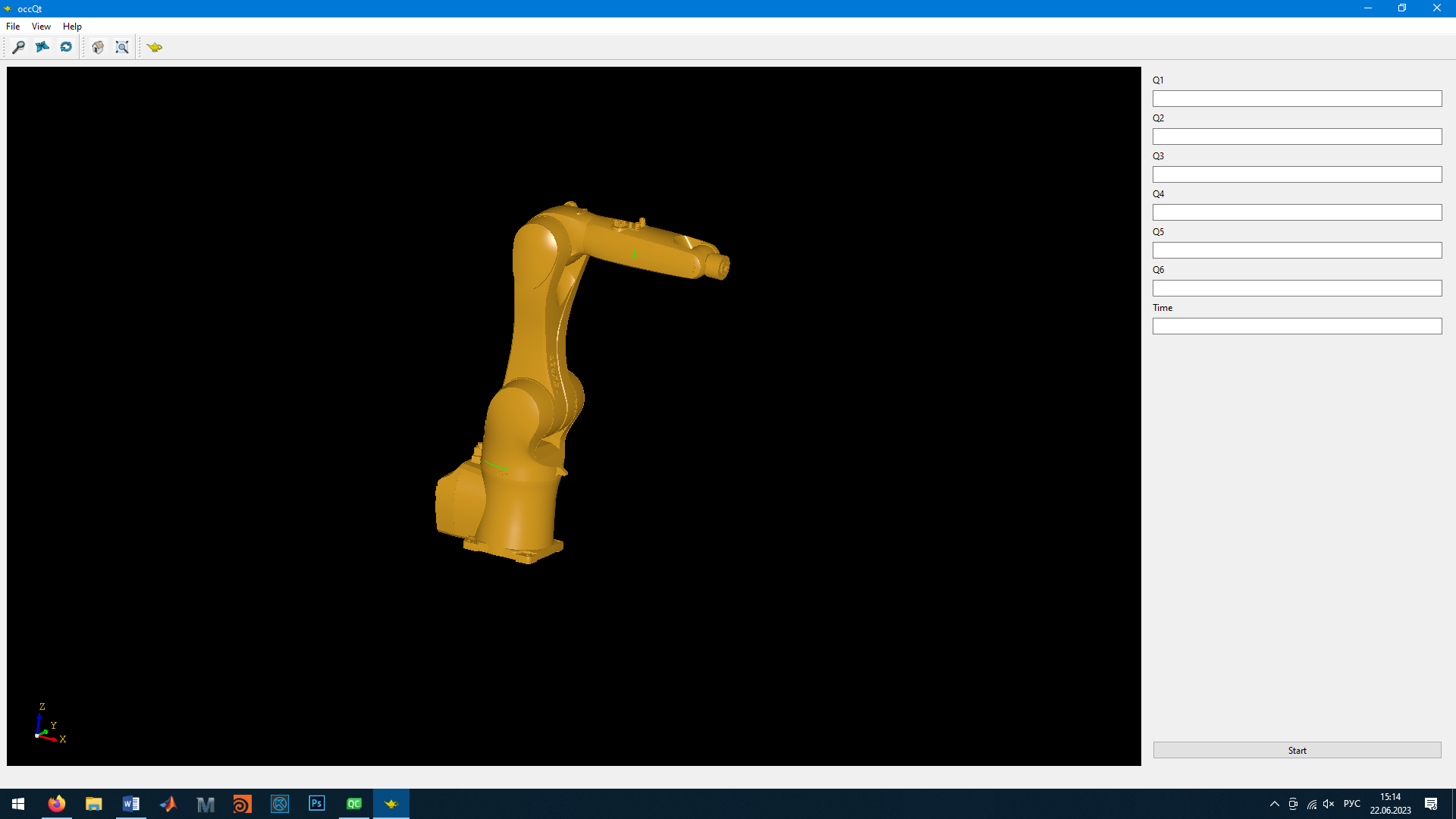


Рисунок 9. Начало анимации.

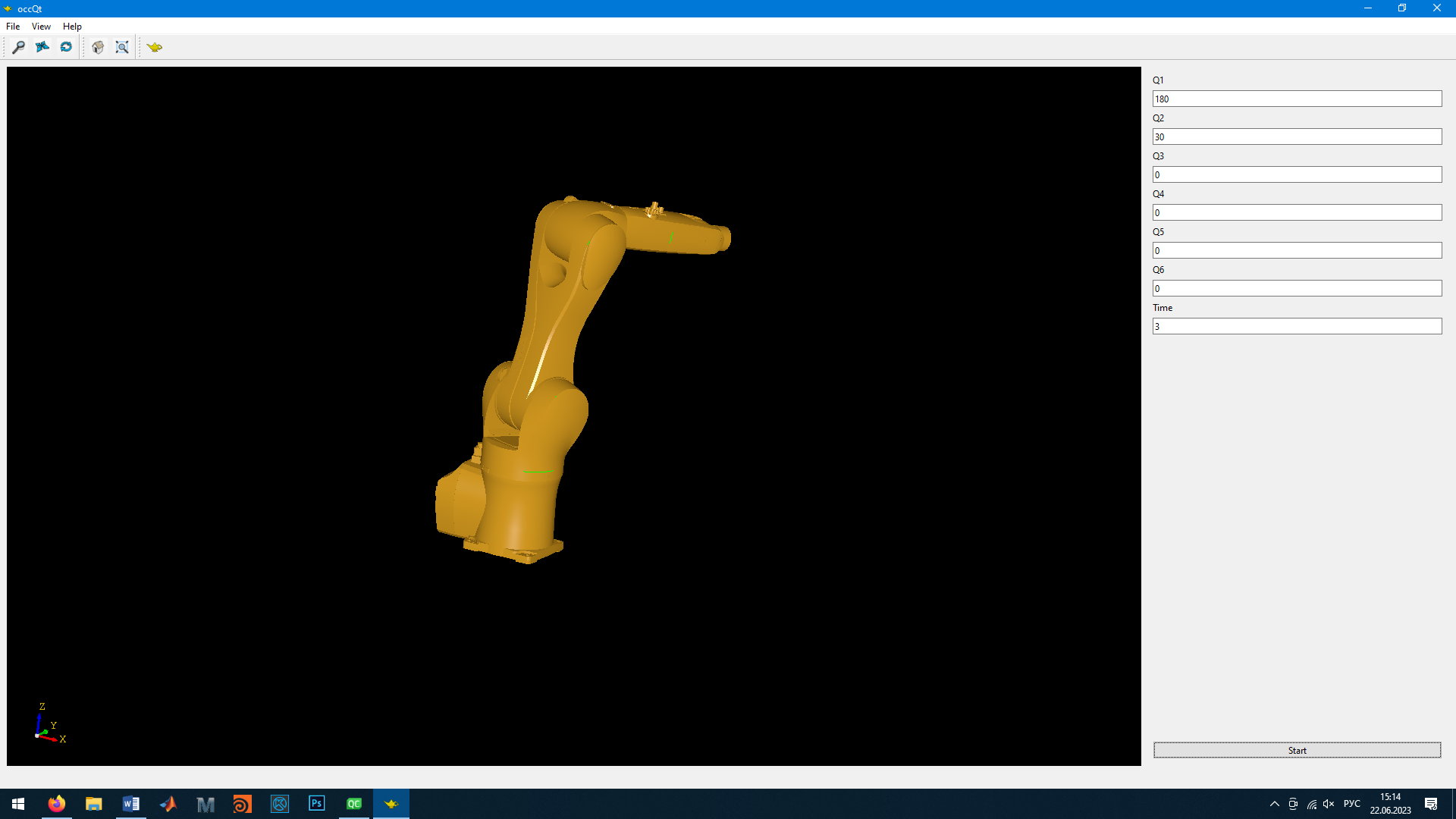


Рисунок 10. Середина анимации.

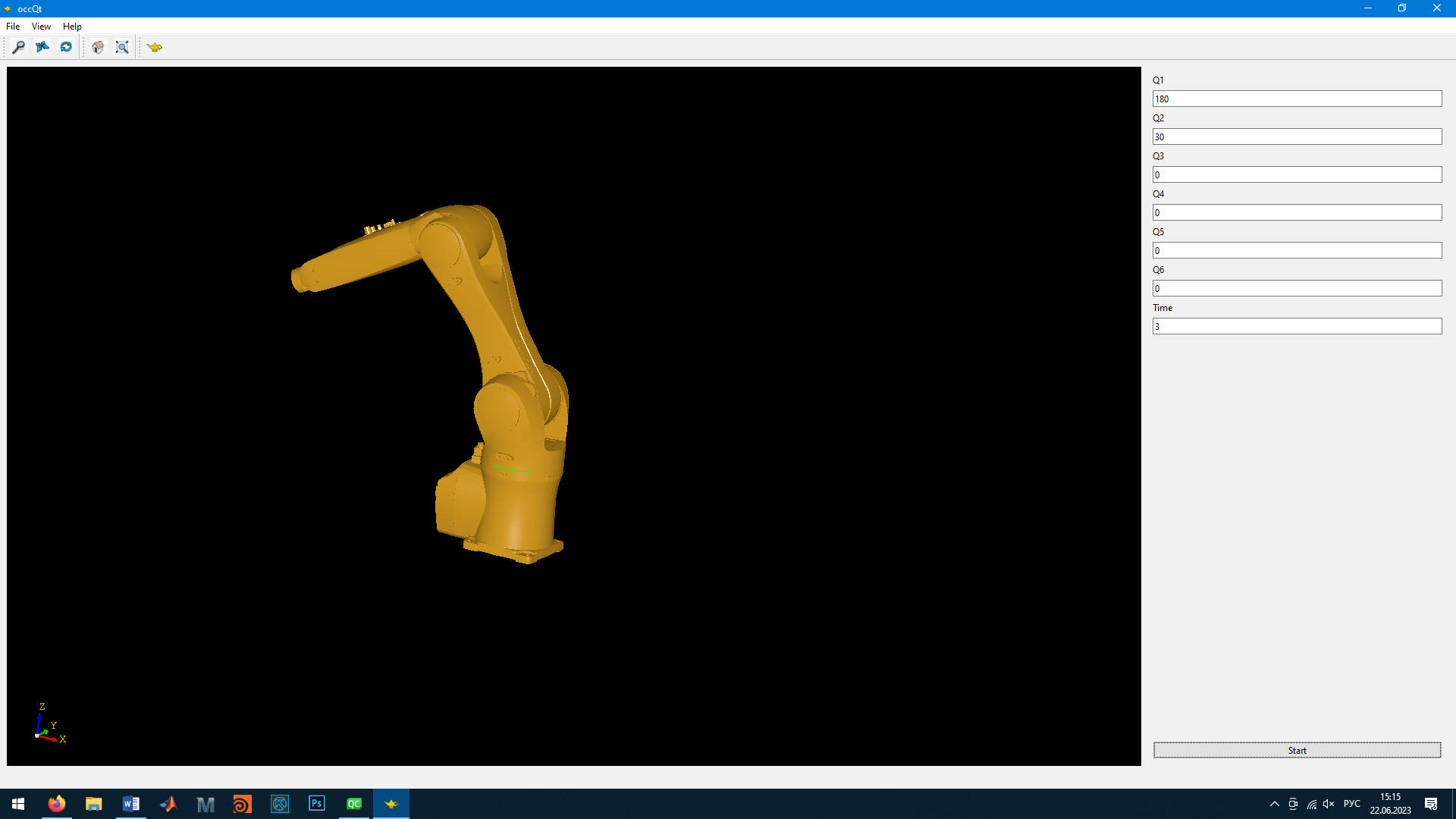


Рисунок 11. Конец анимации.

# **Заключение**

В дальнейшем необходимо провести рефакторинг кода для повышения качества кода и эффективности работы программы. В дальнейшем предполагается разработка иерархии отображаемых step моделей, для лучшего описания связи моделей между собой, не используя громоздкое и ресурсозатратное перемножение матриц преобразование. Предполагается разработать единый API модуля для внешнего приложения управления РТК.

# **Список литературы**

1. Рейтинг языков программирования 2023. – URL: <https://habr.com/ru/articles/730954/> (дата обращения: 19.05.2023). – Текст: электронный.
2. «Дело было вечером, делать было нечего» или краткая история о сравнении производительности языков программирования. – URL: <https://habr.com/ru/articles/563078/> (дата обращения: 19.05.2023). – Текст: электронный.
3. Пишем программу 3D-моделирования в 500 строках кода. – URL: <https://habr.com/ru/articles/527914/> (дата обращения: 20.05.2023). – Текст: электронный.
4. Описание продуктов фирмы C3D LABS. – URL: <https://c3dlabs.com/ru/products/c3d-toolkit/> (дата обращения: 22.05.2023).
5. Open Cascade Technology. – URL: <https://ru.wikipedia.org/wiki/Open_Cascade_Technology> (дата обращения 22.05.2023).
6. Документация библиотеки OpenCascade. – URL: <https://dev.opencascade.org/doc/overview/html/> (дата обращения 25.05.2023).
7. С++. Практика многопоточного программирования.